Projeto Final - RISC-V Multiciclo - OAC, turma C

Nome: Luiz Carlos Schonarth Junior

Matrícula: 19/0055171

Universidade de Brasília - UnB

Introdução

Esse projeto tem como objetivo implementar a arquitetura RISC-V multiciclo, aplicando os conhecimentos sobre *datapaths* e máquinas de estado. Nesta arquitetura, cada etapa da execução de uma instrução dura um ciclo do relógio. As etapas são governadas pelo controle do processador que, no escopo desse projeto, foi implementado usando uma máquina de estados finita.

Implementação do processador

- **Instruções implementadas**

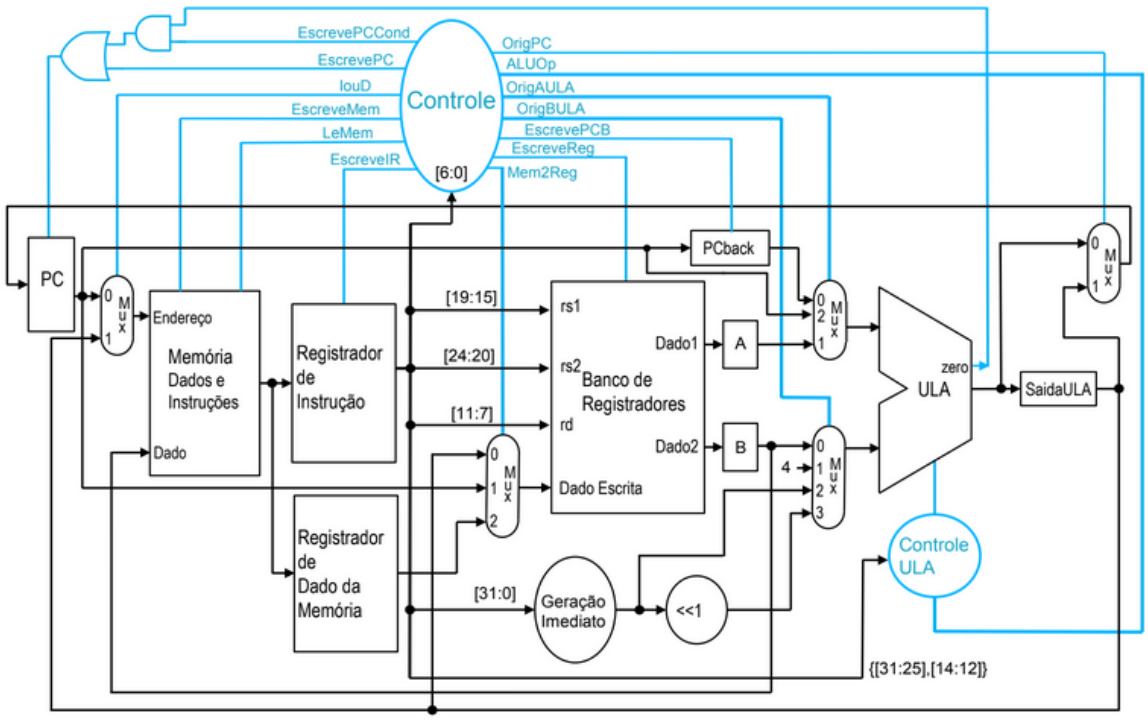
  Foram implementadas as seguintes instruções do *instruction set* RV32I:

Instrução	Formato	Nome da instrução
lw	I	Load Word
sw	S	Store Word
add	R	Add
addi	I	Add Immediate
sub	R	Subtract
and	R	AND
or	R	OR

xor	R	XOR
slt	R	Set Less Than
jal	J	Jump & Link
jalr	I	Jump & Link Register
lui	U	Load Upper Immediate
auipc	U	Add Upper Immediate to PC
beq	B	Branch if Equal
bne	B	Branch if Not Equal
bge	B	Branch if Greater or Equal
bgeu	B	Branch if Greater or Equal Unsigned
blt	B	Branch if Less Than
bltu	B	Branch if Less Than Unsigned

- **Descrição da implementação**

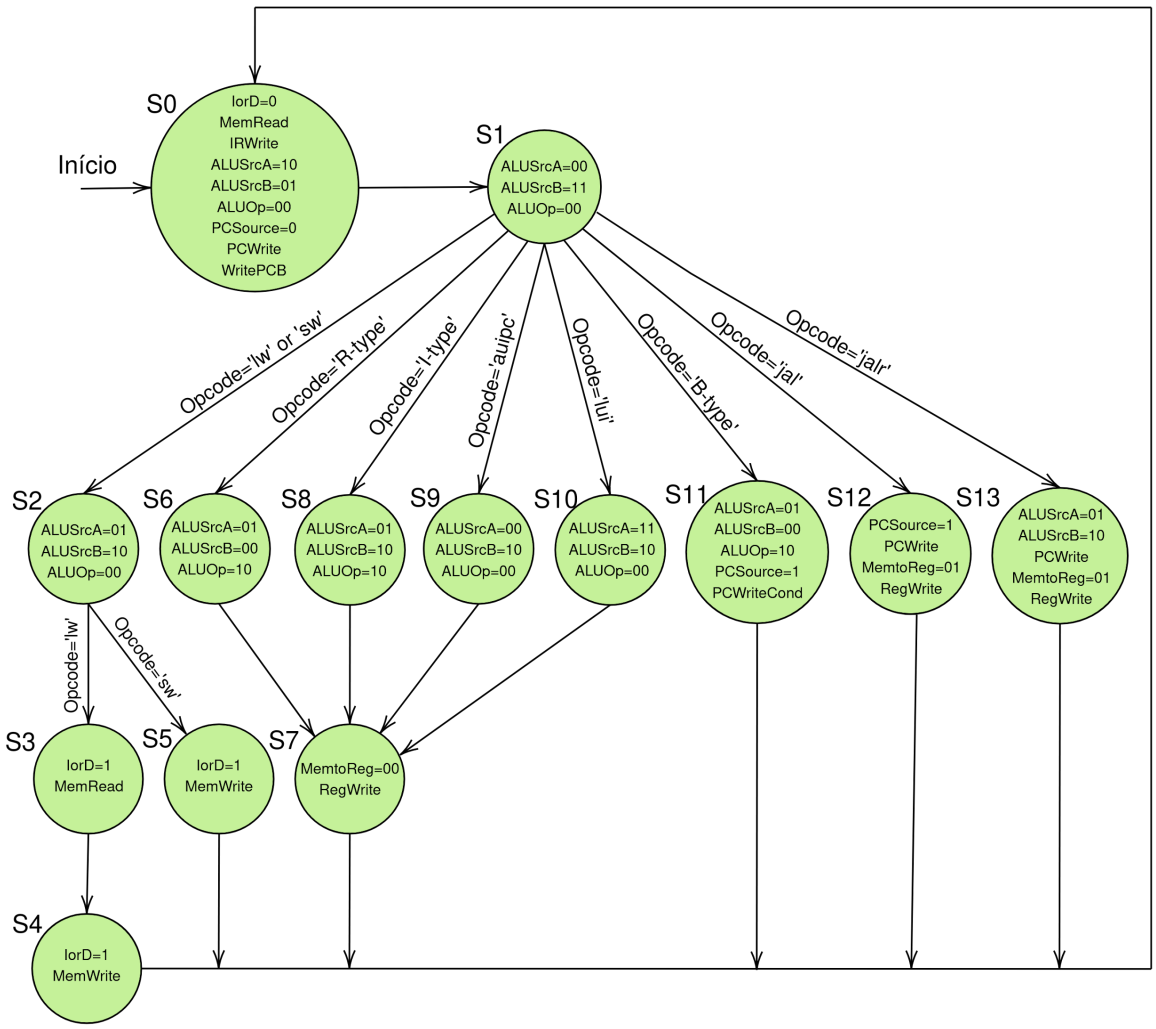
Foi usada a linguagem VHDL de para descrição do *hardware* referente ao processador. o diagrama a seguir ilustra melhor a arquitetura sendo implementada:



A implementação multiciclo conta com registradores a mais, para que os dados obtidos a cada etapa da execução de uma instrução não seja sobrescrita durante a etapa seguinte. Esses registradores são o registrador de instrução (IR) , o registrador de dado da memória (MDR) e os registradores depois banco de registradores e ULA.

As etapas da execução de uma instrução são governadas pela unidade de controle do processador por meio de estados. Como o número de etapa varia entre instruções, ou seja, duram um número de ciclos diferentes, são necessários sinais de controle para definir, por meio de multiplexadores, o caminho dos dados requerido para implementar cada etapa de uma determinada instrução. Esses sinais são as saídas da unidade de controle e são definidas pelo estado atual do controlador, bem como pelo *opcode* da instrução atual.

A seguir pode-se observar o diagrama de estado implementado neste projeto, contendo as saídas/entradas referentes a cada estado:



No pior dos casos, a execução de uma instrução deve passar por 5 etapas (estados do controle), onde essas etapas são: **busca da instrução (S0), decodificação/leitura dos registradores (S1), execução/cálculo do endereço (S2, S6, S8, S9, S10, S11, S12, S13), acesso à memória/conclusão tipo-R e tipo-I (S3, S5, S7)** e **conclusão lw (S4).**

Durante alguns estados do controle, são necessárias operações de adição na ULA, como por exemplo no incremento do PC *(program counter),* cálculo do endereço de *branch/jump* e cálculo do endereço de acesso à memória. Para controlar essas adições e, simultaneamente, suportar as operações de tipo-R e tipo-I, é necessário um controle para definir à ULA qual operação será feita.

O sinal ***ALUOp*** do controle do processador indica ao controle da ULA se uma operação de *ADD* deve ser feita (ALUOp=00), ou se a operação é definida pelos campos *func3* e *func7* da instrução (ALUOp=10). A saída do controle da ULA carrega a informação de qual operação será executada.

Por meio desses controles e registradores a mais, é possível implementar o *set* de instruções do RV32I usando uma arquitetura multiciclo.

- **Dificuldades encontradas**

Uma das dificuldades da implementação é o fato do diagrama das conexões dos módulos da arquitetura, visto na seção anterior, não implementar algumas das instruções que foram implementadas nesse projeto, como ***lui, auipc, jalr,*** entre outras.

Particularmente, no caso da implementação instrução ***lui***, foi adicionado um sinal *zero* ao multiplexador conectado a entrada A da ULA, alterando o número de entradas do multiplexador de três para quatro; dessa forma, é possível adicionar o imediato da instrução com o valor 0 e inserir o resultado no registrador de destino. Essa implementação requer um novo estado no controle do processador, uma vez que é preciso alterar o valor de ALUSrcA de forma a usar o valor *zero*.

A implementação do módulo da memória interna do RISC-V também causou dificuldades, já que o endereçamento das instruções é feita *byte por byte*, porém a implementação da memória utiliza um vetor de palavras de 32 bits (4 bytes), o que requer tratamento dos valores usados para acesso à memória, dividindo o valor por quatro.

Além disso, a implementação da instrução ***jalr*** requer que o bit menos significativo do imediato seja 0, apesar de ser uma instrução do tipo-I, o que pode gerar confusão.

Testes realizados

Para testar a arquitetura montada, foi utilizado um arquivo *assembly* para teste, contendo todas as instruções implementadas neste projeto. Foi utilizado o *software* RARS para gerar os arquivos hexadecimais contendo as instruções e os dados do código.

Os arquivos hexadecimais gerados, juntamente com o código fonte do programa teste.